BEST: An Efficient Algorithm for Mining Frequent Unordered Embedded Subtrees

Israt J. Chowdhury and Richi Nayak

School of Electrical Engineering and Computer Science, Science and Engineering Faculty,
Queensland University of Technology, Brisbane, Australia
{israt.chowdhury, r.nayak}@qut.edu.au

Abstract. This paper presents an algorithm for mining unordered embedded subtrees using the balanced-optimal-search canonical form (BOCF). A tree structure guided scheme based enumeration approach is defined using BOCF for systematically enumerating the valid subtrees only. Based on this canonical form and enumeration technique, the **b**alanced optimal search **e**mbedded **s**ub-**t**ree mining algorithm (BEST) is introduced for mining embedded subtrees from a database of labelled rooted unordered trees. The extensive experiments on both synthetic and real datasets demonstrate the efficiency of BEST over the two state-of-the-art algorithms for mining embedded unordered subtrees, SLEUTH and U3.

Keywords: Frequent subtrees, labelled rooted unordered trees, embedded subtrees, canonical form, enumeration approach.

1 Introduction

The problem of finding frequent subtrees from the tree structured data has important applications in diverse areas including web mining, XML mining, computer vision, network routing and bioinformatics. From the tree structured data, frequent subtree mining discovers important patterns in the tree form showing the distinct features of the data. For example, in [1] frequent subtree mining is used in web log data to distinguish users according to their browsing behaviors on web. It also facilitates other data mining tasks such as association rule mining, classification and clustering.

The tree structured data is often represented in ordered form in which parent and siblings relationships (i.e., fixed left-to-right order) are preserved. However, in practice, the ordering among siblings is not always of great importance to users and is not always available [2]. Unordered trees have shown the capability of identifying interesting relations due to not being constrained by sibling conditions [3, 4]. This distinct property of unordered trees, however, makes the process of mining frequent subtrees more challenging in comparison to ordered trees. A huge number of candidate generation occurs where subtrees with similar structure are included. Besides, it is non-trivial to determine the "good" growth strategy and avoid redundancy, as there can be many possible ways to extend an existing pattern in a tree format, due to not having

an order constraint in sibling nodes. Moreover, high computational and memory expense are an ongoing issue for mining tree structured data.

Two possible types of subtrees, Induced and Embedded, can be mined from the tree data, preserving parental and ancestral relationships respectively. Mining embedded subtree can be seen as a generalization task of mining induced subtree that is essential to mine interesting relational information inherent within deeply embedded data objects in the tree database. It is a more difficult problem than induced subtree mining as it requires examining several levels within a tree to identify an embedded subtree [5].

In this paper we present an algorithm for mining unordered embedded subtrees. Distinct from existing tree traversal methods [6], we have previously proposed an optimal tree traversal algorithm for traversing a rooted unordered tree without enforcing an order among sibling nodes [7]. We extended this traversing algorithm by introducing a new heuristic that leads towards a new definition of canonical form for representing unordered trees, called the balanced-optimal canonical form (BOCF) [8]. The BOCF is able to represent unordered trees uniquely even in the presence of isomorphism.

In this paper we study some properties of the BOCF and design an optimal enumeration tree using BOCF that systematically enumerates all frequent embedded subtrees based on the tree structure guided scheme. This enumeration approach is efficient as it restricts the search by only generating the unambiguous and valid subtrees using the underlying tree structure information. For growing the enumeration tree as well as generating candidates, we define extension and join operations. Finally, the **b**alanced optimal search **e**mbedded **s**ub**t**ree miner algorithm (BEST) is proposed for mining embedded subtrees from a database of labelled rooted unordered trees. Empirical analysis carried out using both real and synthetic data has shown the effectiveness of BEST over the two state-of-the-art algorithms, SLEUTH [5] and U3 [9].

2 Related Works

For finding unordered frequent tree patterns, most of the proposed algorithms use a canonical form and extend only candidates that are in the canonical form. A sorted pre-order string canonical form that can be obtained in linear time was first defined by [10]. A few more similar canonical representations based on depth-first traversal and breadth-first traversal have been defined [11-13]. The proposed method BEST uses the optimal traversal based canonical form (BOCF) that is robust to isomorphism problem due to its order independence and use of optimization. Using BOCF, we proposed a tree structure guided scheme based enumeration technique that uses both right-path extension and join to grow for mining unordered embedded subtrees. None of the above state-of-the-art methods used similar structure guided enumeration process. HybridTreeMiner uses extension and join operations for growing the enumeration tree like BEST using the BFCF canonical form, but for mining induced subtrees. Whereas, SLEUTH [5] is designed to mine embedded subtrees and also uses extension and join operations for growing the enumeration trees but the join is scope-list

join via the descendant and cousin tests. More recent methods UNI3 [14] and U3 [9] also proposed a tree model guided enumeration where they used embedded level information, but we incorporated much more tree information including level, fan-out and a new tree parameter called weight for proposing the tree structure guided enumeration. Moreover they used only right path extension for growing the enumeration tree and used depth-first traversal based string representation which requires additional processing for tackling isomorphism. The unordered embedded subtrees [15, 16] mining algorithm, Treefinder, can miss some patterns especially for a lower support and others have been designed for mining maximal embedded subtrees [15, 16].

3 Mining Embedded Frequent Subtrees

We present the balanced-optimal canonical form, BOCF. We describe the tree structure guided scheme based enumeration approach and the proposed BEST algorithm.

3.1 Preliminaries

Unless otherwise stated, all trees considered in the paper are rooted, labelled, and unordered. Let $T = (V, E, L)$ be a *rooted labeled unordered* tree, where $V = \{v_0, v_1, v_2, \ldots, v_n\}$ denotes the set of nodes with v_0 as *root* node, $E = \{(v_i, v_j) | v_i, v_j \in V\} = \{e_1, e_2, \ldots, e_{n-1}\}$ denotes the set of edges and L denotes the set of labels. The label is given by a function Φ: $V \rightarrow L$ which maps nodes with unique labels. The size of a tree is denoted as $|T|$ which is the number of nodes $|V|$. An unordered tree has no ordering relationship among the nodes except ancestor-descendent or parent-child. The ancestor-descendent relationship between two nodes is denoted by $v_i \prec v_j$, i.e., v_i is ancestor of v_j, the '$\prec$' symbol represents 'precedes'. The level of a node v_i in a tree T is denoted as $Lv(T, v_i)$ and the height of a tree T is denoted as $H(T)$.

Definition 1 (*Embedded Subtrees*): A tree $T'(V', L', E')$ is an unordered *embedded subtree* of a tree $T(V, L, E)$ iff: (1) $V' \subseteq V$, (2) $E' \subseteq E$, (3) $L' \subseteq L$ and the labelling of V' in T is preserved in T' (4) $\forall v_i' \in V'$, $\forall v_i \in V$ and v_i' is not the root node, then ancestor of v_i' = ancestor of v_i, and (5) no left-to-right ordering among the siblings in T is preserved among the corresponding nodes in T'.

Definition 2 (*Equivalent Node*): In a rooted labelled unordered tree T, if two nodes v_i and v_j have the same label ($lab_i = lab_j$ & $lab_i, lab_j \in L$), originated from the same labelled parent node (parent of v_i = parent of v_j) and has the same labelled child nodes then they are called *equivalent nodes*, denoted by $v_i \cong v_j$.

Definition 3 (*Weight of Node*): *Weight* of a node v_i ($v_i \neq v_0$) is defined as the total number of its equivalent node. For tree T, weight of node v_i is w_i such that w_i = total number of equivalent nodes of v_i.

Definition 4 (*Mining Unordered Embedded Subtree*): Let T_{db} is a database, where each transaction is a labelled rooted unordered tree. The task of mining frequent un-

ordered embedded subtree from T_{db} is finding all embedded subtrees that have minimum support *s*.

Definition 5 (*Support*): Support *s* of a tree T' in T_{db} is defined as the number of trees, *T* that has at least one occurrence of T' as an embedded subtree in its structure.

3.2 Balanced Optimal Canonical Form (BOCF)

We first describe the balanced optimal canonical form (BOCF) for a rooted ordered tree [7, 8]. A canonical form (CF) of a tree is a representative form that can consistently represent many equivalent variations of that tree into one standard [6, 13]. The canonical forms for ordered and unordered subtrees are different. A main difference is the possibility of having several subtrees showing different orders between sibling nodes, even though, the information contained within the structure remains essentially the same. Several ordered variations can be formed from a unique unordered tree. This leads us to define Equivalent ordered trees [8].

Definition 6 (*Equivalent ordered trees*): Two distinct ordered trees T_1 and T_2 are equivalent to each other if they represent same unordered tree *T*, denoted by $T_1 \cong T_2$.

An example of equivalent ordered trees is given in fig 1, where four rooted ordered trees can be derived from a rooted unordered tree. We propose to represent these ordered variations by a single canonical form following the optimal tree traversing so that the same unordered tree is derived from each of them.

The canonical form, BOCF is defined by using the order obtained by traversing the tree optimally [7]. BOCF is a string representation of a tree that records label of each node along with its weight following the optimal order [7, 8]. This string also includes four unique symbols, +1, -1, +2 and -2, to represent the breadthwise movement from sibling to sibling and depth-wise movement from a child to its parent. The symbols +1 and -1 are used for depth-forward and depth-backward travel respectively. The symbols +2 and -2 are used for breadth-forward and breadth-backward travel respectively. It is assumed that the alphabet of node labels includes none of these symbols.

An Example: In fig 1 the string encoding using BOCF of the four ordered trees are (a) "$0v_a$, +1, $2v_c$, +1, $2v_d$, -1, -2, $1v_e$, +1, $1v_d$, -2, $1v_f$"; (b) "$0v_a$, +1, $2v_c$, +1, $2v_d$, -1, -2, $1v_e$, +1, $1v_d$, +2, $1v_f$"; (c) "$0v_a$, +1, $2v_c$, +1, $2v_d$, -1, +2, $1v_e$, +1, $1v_d$, -2, $1v_f$"; (d) "$0v_a$, +1, $2v_c$, +1, $2v_d$, -1, +2, $1v_e$, +1, $1v_d$, +2, $1v_f$".

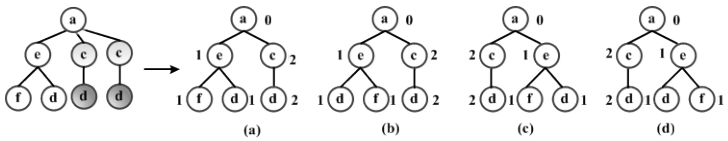

Fig. 1. Four rooted ordered trees obtained from the same rooted unordered tree. Different equivalent nodes are shown as highlighted; weights of nodes are calculated accordingly.

We prove that there exists a one-to-one correspondence between a labelled rooted *ordered* tree and its BOCF.

Lemma 3.1: *Each labelled rooted ordered tree corresponds to a unique balanced optimal canonical form. Each valid balanced optimal canonical form corresponds to a unique labelled rooted ordered tree.*

PROOF: *Since the traversing path of a tree is determined using an optimization model, each ordered tree from an equivalent group for that unordered tree actually represents the same network. Consequently, the optimal traversal gives the same traversing order to all equivalent ordered trees. BOCF is defined using this optimal order along with some unique symbols to capture the sibling constraints for the different ordered trees. As a result, each labelled rooted ordered tree will be represented by a unique BOCF.*

The second statement of the aforementioned lemma is proved by the induction on the number of nodes N in a labelled rooted ordered tree. For the base case, when N = 1, the valid string representation of BOCF is of the form 0lab$_i$, where lab$_i$ (lab$_i$ ∈ L) is the label of the single node v_i; weight 0 indicates a root node. In this case, the corresponding labelled rooted unordered tree is a single node, which is unique.

For simplicity of this proof we group all unique symbols of representing the sibling constraints; let C be the group containing all the unique symbols for representing constraints where C ∉ L and {-1, +1, -2, +2} ∈ C. So incorporating this notation the string representation, S of BOCF can be represented as "S = "w_0, lab$_0$, C, w_i, lab$_i$, …". For the induction step, we assume that, for each BOCF string representation S_n with N = n nodes, there is a unique labelled rooted ordered tree in corresponding to it. A valid BOCF string representation S_{n+1} with N = n + 1 nodes is of the form "S_n . . . C, w_{n+1}, lab$_{n+1}$". S_n determines a unique labelled rooted ordered tree with n nodes. In addition, the last node (with label lab$_{n+1}$) becomes the rightmost child of node n. As a result, the labelled rooted ordered tree N_{n+1} corresponding to S_{n+1} is determined uniquely.

Consider the example in fig 1, for a rooted unordered tree, different rooted ordered trees and the corresponding BOCFs are obtained by assigning different orders among the children of internal nodes. The BOCFs of equivalent ordered trees only vary in terms of breadth movement, which shows the order of siblings for different trees that can be ignored for portraying the unordered tree. The BOCF string representation of the rooted unordered tree is defined by a guided breadthwise movement while forming the string of ordered trees. The rest of the ordering that reflect ancestor descendent relationship is kept unchanged.

Definition 7 (*BOCF String Representation of Unordered Tree*): The BOCF string representation of the rooted unordered tree is achieved by a guided record of sibling node. When a new node is recorded under its parent node, only the breadthwise movement from the existing rightmost sibling node is permitted.

By doing so, all equivalent ordered trees will be represent by a unique standard form, which will be advantageous for unordered tree mining. Consider again the example of fig 1, using definition 7 the string representation of all four equivalent ordered trees are: (a) "0v_a, +1, 2v_c, +1, 2v_d, -1, +2, 1v_e, +1, 1v_d, +2, 1v_f"; (b) "0v_a, +1, 2v_c, +1, 2v_d, -1, +2, 1v_e, +1, 1v_d, +2, 1v_f"; (c) "0v_a, +1, 2v_c, +1, 2v_d, -1, +2, 1v_e, +1, 1v_d, +2, 1v_f"; (d) "0v_a, +1, 2v_c, +1, 2v_d, -1, +2, 1v_e, +1, 1v_d, +2, 1v_f", which are same and represent the fact that they are originated from the same unordered tree.

Lemma 3.2: *The BOCF construction procedure for unordered trees has time complexity O* (|*T*| *log* |*T*|).

PROOF: The *optimal traversal algorithm gives O*(|*T*| *log* |*T*|) *time complexity where* |*T*| *is the number of nodes in a tree. Implementing any of the three heuristics [8] of optimal traversal for sorting nodes will give a possible time complexity of O*(|*T*| *log* |*T*|*). Assuming there are* $|T_j|$ *nodes in recursion j of the tree traversal for j =1, 2, …n, it will take O*($|T_j|$ *log* $|T_j|$) *comparisons to sort nodes at recursion j. The total number of comparisons for normalizing the whole tree is* $\sum_j O(|T_j| \log |T_j|)$*, which is O*(|*T*| *log* |*T*|) *(note that* $\sum_j (|T_j| \log |T_j|) \leq \sum_j (|T_j| \log |T|) = |T|$ *log* |*T*|). BOCF is driven using the exact ordering of optimal traversal, therefore its construction complexity is also *O* (|*T*| *log* |*T*|).

It can be noted that all equivalent ordered trees is represented by a unique standard form and indicate that they are originated from the same unordered tree. This greatly benefits unordered tree mining. The optimal traversal poses a total order on all variants of the same unordered tree which guarantees the uniqueness of BOCF for a labelled rooted *unordered* tree.

Handling the Isomorphism and Automorphism Problems: Two trees T_1 and T_2 are isomorphic to each other if a bijective mapping exists between their sets of nodes, which preserves and reflects their structures, denoted as $T_1 \cong T_2$. If isomorphism exists within a tree, then it is called automorphism. It is necessary to identify which of the ordered subtrees belongs to an automorphism group of an unordered subtree in order to ensure the exact count of its occurrences as well as the frequency. Therefore, canonical form should be defined in a way that will uniquely map each subtree to a single subtree during candidate generation. Existing research addresses this problem by choosing one of the trees from the automorphism group as the representative of the group, and then all other isomorphic subtrees are ordered according to the representative of the automorphism group during candidate generation [5]. However, a checking is always required to find the presence of isomorphism in a tree, which causes additional memory consumption for keeping the record of the representative tree during the candidate generation phase, thus, the exact ordering can be followed for generating other isomorphic subtrees.

Proposed BOCF addresses this problem [8] as follows. It gives a unique representation to all isomorphic trees without requiring any representative tree record or, any

extra checking during candidate generation. Moreover, it naturally handles the automorphism problem by using the concept of weights (definition 3) to represent equivalent nodes (definition 2). The equivalent nodes for an unordered tree should not be treated distinctively since their occurrences are important for mining, not the inherent ordering between sibling nodes. Consider the following example where the dotted area shows a case of automorphism problem for the considered tree. The proposed canonical form is derived based on the weighted tree as shown in fig 2(b) where automorphism can no longer exist.

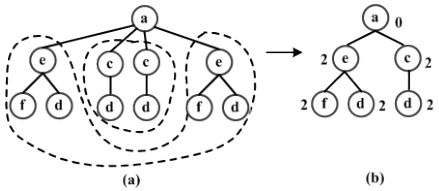

Fig. 2. Automorphism problem

3.3 The Enumeration Tree

In this section we define an enumeration tree that enumerates all embedded unordered subtrees in T_{db} according to their BOCFs. We used both right-path extension and join operation for growing the enumeration tree. Previous research has shown that the right-path extension produces a complete and non-redundant candidate generation [17]. Due to the large number of potential growth, only using extension for growing an enumeration tree can be inefficient, especially when the cardinality of the alphabet for node labels is large [5, 13]. This emphasizes the need of using a join operation; however, it often generates invalid subtrees. Since we use a tree structure guided scheme for enumeration, this generates valid subtrees only.

Tree Structure Guided Scheme Based Enumeration: This enumeration is a bottom-up approach that generates non-redundant candidates [18]. A candidate generation technique can generate valid frequent and infrequent candidates as well as invalid frequent and infrequent candidates. It is desirable to enumerate valid frequent subtrees only to save memory and computational expense, instead of generating all possible candidates and prune invalid and infrequent subtrees later.

To illustrate this, we show a simple tree structure as an example database in fig 3. We also show some possible valid and invalid subtrees that can be generated from this example tree. The subtree that does not follow the available tree structure information (i.e., the position of various nodes at various levels, ancestor-descendent or parent-child relationship, number of child nodes under parent node, etc.) is considered invalid. In our proposed tree structure guided scheme based enumeration, we utilize underlying level and fan-out information of nodes during candidate generation to make the approach structure guided. For efficiently growing the enumeration tree we define the extension and join operations using BOCF and the tree structure guided scheme.

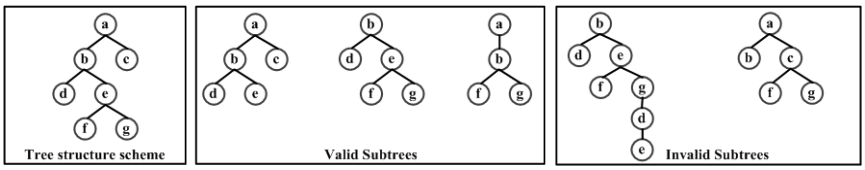

Fig. 3. Valid and invalid subtrees following tree structure guided scheme.

Definition 8 (*Extension*): From a node v_i (fan-out $\neq 0$) of the BOCF tree T_1, extension is possible by adding a frequent label v_j having a level $> Lv(T_1, v_i)$. This will result in a new BOCF tree T_2 in the enumeration tree where v_j will be the child of v_i. If T_1 is a N-tree then the resultant new BOCF tree T_2 will be a (N+1)-tree with a height $H(T_1)$+1. Further extension will be possible from this newly added right-most node v_j.

Before giving the definition of join operation, we define equivalent groups.

Definition 9 (*Equivalent group*): If two N-node trees T_1 and T_2 have height $H(T_1) = H(T_2)$ and have the first N-1 nodes (along with labels and weights) common, they are considered as equivalent group, denoted by $T_1 \cong T_2$.

Definition 10 (*Join*): Join operation is a guided extension between two BOCF trees T_1 and T_2 from an equivalent group, $T_1 \cong T_2$. Assume v_i and v_j are the corresponding right-most nodes of T_1 and T_2 respectively, where $w_i > w_j$ or $w_i = w_j$ with v_i lexicographically sorts lower than v_j. By joining v_j in T_1 at the position of $Lv(T_1, v_i)$-1 will result in a new (N+1) node BOCF tree, denoted $T_1 \odot T_2$, of the same height as BOCF tree T_1.

Growth Rules: Candidate trees can have a large number of potential nodes to get a right-path extension. In order to restrict this growth, heuristics can be employed. This will result in reduction of the number of candidates generated as well as in the reduction of the number of isomorphic subtrees. These rules support the basic formation principle of the enumeration tree, i.e., keeping the N-tree BOCF unchanged with the newly generated N+1- tree BOCF.

Rule1: *Among all the nodes at the bottom level, the node that has the maximum weight will be chosen for applying an extension.*

Rule2: *If there are more than two maximum weighted nodes then the node that has the maximum children will be chosen for applying an extension.*

Rule3: *If more than two maximum weighted nodes exist with the same number of children then the node that appears lexicographically lower will be chosen for applying an extension.*

An Example: We compare the enumeration tree generated by BEST with another enumeration tree generated by SLEUTH [5] using an example database in fig 4a. Considering all labelled nodes as frequent, the SLEUTH enumeration tree grows as fig 4c, where the extension and join operations are defined using another canonical form (fig 4c) and are not following tree structure guided scheme. In fig 4b, the pro-

posed BOCF and the tree structure guided scheme based BEST enumeration tree is shown, which is the complete enumeration tree for the given database, whereas the state-of-the-art enumeration tree cannot be completed due to limited space. If we continue, it will grow more. The dotted rectangles in fig 4c show an example of generated invalid subtrees in SLEUTH. Fig 4c only shows some, a lot more is generated during the process, whereas no invalid subtree is generated by BEST. It can be noted that the BEST enumeration tree generates much less candidate trees in comparison to SLEUTH because the former only produces valid subtrees. Consequently, a lot of memory space and additional computational time can be saved that will be required to prune these invalid subtrees afterwards. Empirical analysis ascertains these claims.

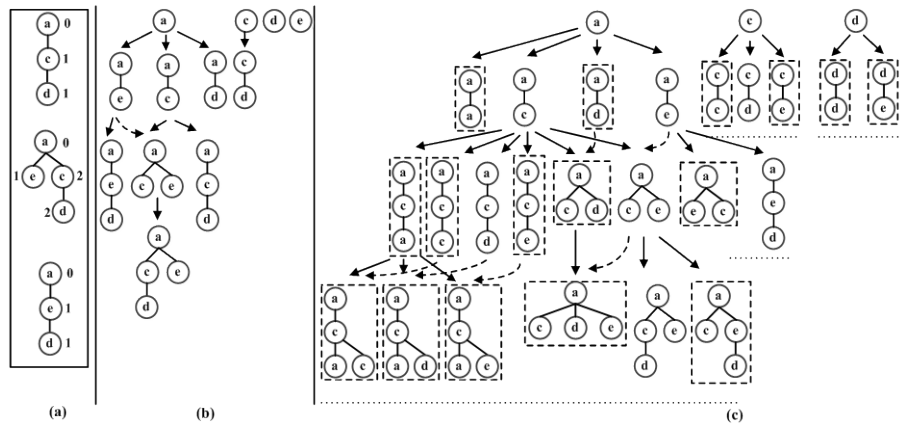

Fig. 4. Comparison between the proposed and an existing enumeration technique.

3.4 The BEST Algorithm

The process of frequent mining is initiated by scanning a database, T_{db}, where trees are stored as BOCF strings along with weight, level and fan-out information of each node. The set of frequent labels (frequent subtrees of size 1) is generated and larger sized subtrees are generated by calling the *Enum* method recursively (fig 5). In *Enum*, a subtree is extended if the right-most node of the tree supports any of the three rules of growing strategy. For implementing extension, the level difference of the right-most node of the considered tree is checked with the frequent label and the new candidate subtree is generated if the condition is met. Frequency of every resultant candidate tree is computed according to the method used in [13]. This is an apriori based frequency counting which gives us the exact frequent subtree list. In order to improve computational efficiency, we stop counting of a subtree as soon as the tree count reaches the minimum support value. Throughout the BEST algorithm the downward-closure lemma [19] is hold; each N-subtree of a frequent N+1-subtree has to be frequent. In the *Enum* function, we also used join for generating candidates from *equivalent groups* that support the join operation and the frequency of each subtree is calculated for further processing.

BEST Algorithm

Input: a database T_{db} consisting of labelled rooted unordered trees in their BOCFs, a dictionary containing level and fan-out information of each node, a user defined minimum support (*min_sup*).
Output: All frequent embedded subtrees.

```
1.  Result ←∅;
2.  F1 ← the set of all frequent nodes;
3.  F2 ← ∅;
4.  while F1 ≠ ∅ do
5.      for all t_k ∈ F1 do
6.          if fan-out(t_k) = 0
7.            continue
8.          end if
9.          Ext_can ← Enum (t_k, level, weight,
            fan-out );
10.         for all t_{k+1} ∈ Ext_can do
11.             if support (t_{k+1}) ≥ min_sup then
12.                 F2 ← F2 ∪ t_{k+1};
13.             end if
14.         end for
15.     end for
16.         F1← F2;
17.         Result ← Result ∪ F1;
18.         F2 ← ∅;
19. end while
20. return Result
```

Enum

Input: candidate C_k, level, weight, fan-out
Output: all (k+1) extensions of C_k

```
1.  out ←∅;
2.  for all frequent label f do
3.    Select the right-most node of C_k using Growth
      rules;
4.    Generate candidate C_{k+1} by adding f;   //using
      definition 8;
5.    out ← out ∪ C_{k+1};
6.  end for
7.  for all C_k´ such that C_k ≅ C_k´ do
8.    C_{k+1} ← C_k ⊙ C_k´;   //using definition 10;
9.    out ← out ∪ C_{k+1};
10. end for
11. return out;
```

Fig. 5. High level pseudo code of BEST algorithm

4 Experimental Evaluation

We have performed extensive experiments to evaluate the efficiency of the proposed BEST algorithm on real application data as well as on synthetic data. All experiments have been conducted on a 2.8GHz Intel Core i7 PC with 8GB main memory and running the UNIX operating system. SLEUTH [5] and U3 [9], used for benchmarking, are designed for mining unordered embedded subtrees and are most relevant to our proposed method.

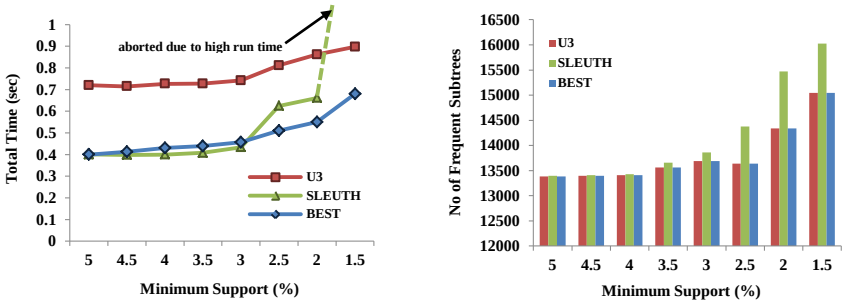

Fig. 6. Comparison over CSLOGS data based on runtime (a) and no of frequent subtrees (b).

Performance on Real Application Data - CSLOGS: In our experiments**,** we used the CSLOGS dataset a real weblog data that consists of 59,691 trees, 716,263 nodes and 13,209 unique node labels [5, 17]. This data set has been largely used to evaluate various frequent subtree mining algorithms [5, 9].

For evaluating the performance we consider the runtime and candidate generation for all three algorithms. For CSLOGS dataset, BEST consistently outperformed SLEUTH and U3 (fig 6(a)). Although SLEUTH performs almost same as BEST, but after a certain value of minimum support (1.5%) it took longer time than the other two algorithms. For SLEUTH the number of candidate subtrees is higher than the other two algorithms, i.e., it includes a lot of invalid subtrees during enumeration, therefore, spends more time on candidate generation and pruning afterwards. Besides, both SLEUTH and U3 require a canonical form test to avoid isomorphism and take longer processing time than BEST.

From fig 6(b), it can be observed that SLEUTH generated more frequent subtrees in comparison to BEST, as it uses the opportunistic pruning technique which does not fulfil the downward closure lemma and may generate pseudo frequent subtrees [18].

Performance on Synthetic Data: Zaki's tree generator [20] is used for generating a synthetic data using following parameters: the number of labels N = 100, the number of vertices in the master tree M = 10,000, the maximum depth D = 10, the maximum fan-out F = 10 and the total number of subtrees T = 100,000. We used three synthetic datasets: D10 had all default values, F5 had all values set to default except for fan-out F = 5, and for T1M we set T = 1,000,000, with remaining default values. These are used for doing scalability and sensitivity analysis.

In fig 7(a) for D10 dataset, U3 performed better than the other two, but the results for U3 are reported here for level difference one, otherwise the algorithm was aborted due to very high memory expense. As we restricted the level difference value to one, so the list of embedding subtrees is not completed and accordingly required less time, whereas both SLEUTH and BEST retrieved all of the embedding subtrees within reasonable time and memory expense.

For F5 dataset, we can see in fig 7(b) BEST outperformed both SLEUTH and U3. Here U3 results are again reported based on restricted level difference, still BEST performed slightly better. Finally for T1M dataset we can see again BEST performed a little better than SLEUTH for lower and higher support values. Again, we only managed to run U3 for extracting embedded subtrees for level difference = 1, hence, it is not reporting the real time for extracting all embedded subtrees.

From these results we notice that both SLEUTH and U3 are sensitive to breadth, for small breadth value (small tree width), these baseline algorithms took high run time, as shown by F5 dataset (the fan-out number is less than D10 and T1M datasets). When SLEUTH and U3 performed over F5, the runtime increased about 8 and 2 times respectively in comparison to runtime over D10 and T1M. BEST seems not sensitive to this parameter and gives a consistent performance. It can be ascertain that BEST is a robust and efficient algorithm in comparison to existing state-of-the-art algorithms for mining embedded subtrees. It can tackle isomorphism using BOCF

canonical form and generates only valid subtrees using the tree structure guided enumeration. These allow BEST to save reasonable amount of time and memory.

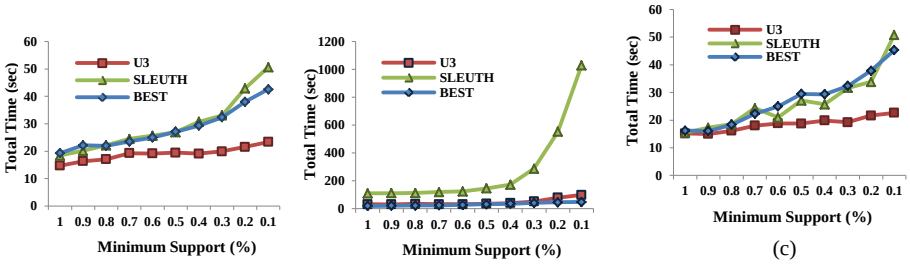

Fig. 7. Comparison over *D*10 (a), *F*5 (b) and *T*1*M* (c) synthetic datasets.

5 Conclusion

In this paper, we presented a novel method for finding frequent embedded subtrees, using an optimal canonical form, from the dataset of labelled rooted unordered trees. We empirically evaluated the efficiency of the proposed method and benchmarked with the well-known algorithms in the literature, over both real and synthetic datasets.

Although finding the condensed representations of frequent patterns has found more interest in recent years, developing efficient algorithms for finding frequent patterns is still important. The efficiency of the algorithms for finding condensed representations depends on the efficiency of the base, i.e., frequent pattern mining algorithms. In future we will extend the proposed algorithm to find condensed representations.

References

1. Zaki, M.J. Aggarwal, C.C.: XRules: An Effective Structural Classifier for XML Data. In: Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 316-325. ACM, Washington, D. C. (2003)
2. Yoshimura, Y. Shoudai, T.: Learning Unordered Tree Contraction Patterns in Polynomial Time. In: Inductive Logic Programming. pp. 257-272. Springer Berlin Heidelberg, (2013)
3. Shasha, D., Wang, J.T.-L., Zhang, S.: Unordered tree mining with applications to phylogeny. In: Proceedings on the 20th International Conference on Data Engineering, (ICDE' 04). , pp. 708-719. IEEE, (2004)
4. Wang, Y., DeWitt, D.J., Cai, J.-Y.: X-Diff: An Effective Change Detection Algorithm for XML Documents. In: Proceedings of the 19th International Conference on Data Engineering, pp. 519-530. IEEE, Vienna (2003)
5. Zaki, M.J.: Efficiently Mining Frequent Embedded Unordered Trees. Fundamental Informatic. 66(1-2), 33-52 (2004)
6. Valiente. Algorithms on Trees and Graphs. Springer, Berlin Heidelberg, New York (2002)

7. Chowdhury, I.J. Nayak, R.: A Novel Method for Finding Similarities between Unordered Trees Using Matrix Data Model. In: Web Information Systems Engineering – WISE 2013. Lin, X., Manolopoulos, Y., Srivastava, D., Huang, G. (eds). pp. 421-430. Springer Berlin Heidelberg, (2013)
8. Chowdhury, I.J. Nayak, R.: BOSTER: An Efficient Algorithm for Mining Frequent Unordered Induced Subtrees (In press). In: Web Information Systems Engineering – WISE 2014. Benatallah, B., Bestavros, A., Vakali, A. (eds). pp. 146-155. Springer Berlin Heidelberg, (2014)
9. Hadzic, F., Tan, H., Dillon, T.S.: U3 - Mning Unordered Embedded Subtrees Using TMG Candidate Generation. In: Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 01, pp. 285-292, IEEE Computer Society. (2008)
10. Luccio, F., Pagli, L., Enriquez, A.M., Rieumont, P.O.: Bottom-up subtree isomorphism for unordered labeled trees. International Journal of Pure and Applied Mathematics. 38(3), 325 (2007)
11. Asai, T., Arimura, H., Uno, T., Nakano, S.-i.: Discovering Frequent Substructures in Large Unordered Trees. Springer Berlin Heidelberg, (2003)
12. Nijssenm, S. Kok, J.N.: Efficient Discovery of Frequent Unordered Trees. In: First International Workshop on Mining Graphs, Trees and Sequences, pp., Springer Berlin Heidelberg. Croatia (2003)
13. Chi, Y., Yang, Y., Muntz, R.R.: Canonical Forms for Labelled Trees and Their Applications in Frequent Subtree Mining. Knowledge and Information System. 8(2), 203-234 (2005)
14. Hadzic, F., Tan, H., Dillon, T.S.: UNI3 - Efficient Algorithm for Mining Unordered Induced Subtrees Using TMG Candidate Generation. In: Proceedings of the 1st IEEE Symposium on Computational Intelligence and Data Mining, pp. 568-575. Honolulu, Hawaii (2007)
15. Chehreghani, M.H., Rahgozar, M., Lucas, C.: Mining maximal embedded unordered tree patterns. In: IEEE Symposium on Computational Intelligence and Data Mining, 2007 (CIDM 2007) pp. 437-443. IEEE, (2007)
16. Termier, A., Rousset, M.-C., Sebag, M.: Treefinder: a first step towards xml data mining. In: IEEE International Conference on Data Mining, 2002 (ICDM 2002), pp. 450-457. IEEE, (2002)
17. Zaki, M.J.: Efficiently mining frequent trees in a forest. In: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 71-80. ACM, Edmonton, Alberta, Canada (2002)
18. Tan, H., Dillon, T.S., Hadzic, F., Chang, E., Feng, L.: IMB3-Miner: mining induced/embedded subtrees by constraining the level of embedding. In: Advances in Knowledge Discovery and Data Mining. pp. 450-461. Springer, (2006)
19. Agrawal, R. Srikant, R.: Fast Algorithms for Mining Association Rules in Large Databases. In: Proceedings of the 20th International Conference on Very Large Data Bases, pp. 487-499. Morgan Kaufmann Publishers Inc., (1994)
20. Zaki, M.J.: Efficiently Mining Frequent Trees in A Forest: Algorithms and Applications. IEEE Transactions on Knowledge and Data Engineering. 17(8), 1021-1035 (2005)